TITLE OF THE INVENTION

LOOP-FREE MULTIPATH ROUTING METHOD USING DISTANCE VECTORS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from U.S. provisional application serial number 60/244,622 filed on October 30, 2000, incorporated herein by reference.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

This invention was made with Government support under Grant No. F19628-96-C-0038 awarded by the Air Force Office of Scientific Research (AFOSR). The Government has certain rights in this invention.

REFERENCE TO A COMPUTER PROGRAM APPENDIX

Not Applicable

NOTICE OF MATERIAL SUBJECT TO COPYRIGHT PROTECTION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention pertains generally to protocols for network traffic routing, and more particularly to a loop-free multipath routing protocol based on distance vectors.

2. Description of the Background Art

Routing protocols using the "Distributed Bellman-Ford" (DBF) algorithm exhibit excessively long convergence process toward correct routes when subjected to link cost increases. A more serious deficiency of the DBF algorithm is that it is unable to converge when a set of link failures result in a network partition, which is commonly referred to as the count-to-infinity problem. Moreover, typical routing protocols utilized for the IP Internet provide a single next-hop choice for packet forwarding. The use of single-hop choices is inadequate for traffic load balancing, while it allows temporary routing loops to form during times of network transition, which diminishes network performance.

Routing may be described as the problem of determining a set of successor choices (i.e., next-hop) at each node and for each destination in the network to be used for packet forwarding. In creating a formal definition, allow a computer network to be represented as a graph $G = (N, L)$, where N is the set of nodes (routers) and L is the set of edges (links). The set of neighbors of node i is to be given by N^i. The problem

consists of finding the successor set at each router i for each destination j, denoted by $S_j^i \subseteq N^i$, so that when router i receives a packet for destination j, it can forward the packet to one of the neighbor routers in the successor set S_j^i. By repeating this process at every router, the packet is expected to reach the destination. If the routing graph SG_j is a directed subgraph of G, as defined by the link set $\{(m, n) | n \in S_j^m, m \in N\}$, a packet destined for j follows a path in SG_j. Two criteria determine the efficiency of the routing graph constructed by the protocol: *loop-freedom* and *connectivity*. It is required that SG_j be free of loops, at least when the network is stable, because routing loops degrade network performance. In a dynamic environment, a stricter requirement is that SG_j be loop-free at *every instant*, such as if S_j^i and SG_j are parameterized by time t, then $SG_j(t)$ should be free of loops at any time t. If there is at most one element in each S_j^i then SG_j is a tree and there is only one path from any node to node j. On the other hand, if S_j^i has more than one element, then SG_j is a directed acyclic graph (DAG) with greater connectivity than a simple tree, and can be utilized to enable traffic load balancing.

The importance of using a successor set instead of a single successor per destination and the need for instantaneous loop-freedom of SG_j has been demonstrated in recent work, in which a load-balancing routing framework is described which obtains "near-optimal" delays. A required key component of this framework is a routing protocol which responds quickly in determining multiple successor choices for

packet forwarding, such that the routing graphs implied by the routing tables are free of loops even during network transitions. By load-balancing traffic over the multiple next-hop choices, congestion and delays are significantly reduced.

A number of limitations exist in the use of current Internet routing protocols. The widely deployed routing protocol RIP provides only a single next-hop choice for each destination and does not prevent temporary loops from forming. A protocol from Cisco™ referred to as EIGRP ensures loop-freedom but can guarantee only a single loop-free path to each destination at any given router. The link-state protocol known as OSPF offers a router multiple choices for packet-forwarding only when those choices offer the minimum distance. When fine granularity exists in the link cost metric, perhaps for the sake of accuracy, it is less likely that multiple paths with equal distance exist between each source-destination pair, which translates to not using the full connectivity of the network for load balancing. Also, OSPF and other similar algorithms which are based on topology-broadcast incur excessive communication overhead, often forcing network administrators to partition the network into areas connected by a backbone. This makes OSPF complex in terms of the required router configurations.

Several routing algorithms based on distance vectors have been proposed within the industry. However, with the exception of DASM (Zaumen, W. T. and Garcia-Luna-Aceves, "Loop-Free Multipath Routing Using Generalized Diffusing Computations", Proc. IEEE INFOCOM, March 1998) which provides multiple loop-free paths per destination, all of the proposed solutions are single-path algorithms. In addition, a number of distributed routing algorithms have been proposed that use the distance and

second-to-last hop to destinations as the routing information exchanged among nodes. These algorithms are often called path-finding algorithms or source-tracing algorithms. One of these path finding algorithms, referred to as LPA appears to provide greater efficiency than any of the routing algorithms based on link-state information proposed to date while it provides loop-freedom at every instant. Again, however, it should be appreciated that LPA along with the other current source-tracing algorithms provide only a single path per destination. A couple of routing algorithms have been proposed that use partial topology information, such as LVA, and ALP, to eliminate the main limitation of topology-broadcast algorithms. These routing algorithms, however, do not provide loop-freedom at every instant.

Recently, MPDA has been introduced, which appears to be the first routing algorithm based on link state information that provides multiple paths to each destination that are loop-free at every instant. Another algorithm referred to as MPATH, has been introduced which appears to be the first path-finding algorithm that constructs loop-free multipaths. Currently MPDA, MPATH, and DASM appear to offer the only practical loop-free multipath routing algorithms which are suitable for implementation within a near-optimal routing framework.

Therefore, a need exists for a routing protocol that allows the construction of loop-free multipaths, even during network transitions, while still providing collision-free communication as outlined above. The present invention satisfies those needs, as well as others, and overcomes the deficiencies of previously developed routing protocols.

BRIEF SUMMARY OF THE INVENTION

The present invention comprises a distance vector routing methodology referred to as a "Multipath Distance Vector Algorithm" (MDVA) that computes the shortest multipath loop-free routes between each source and destination pair. In MDVA, only distance values are exchanged among neighboring routers.

By way of example, and not of limitation, in MDVA, link distances D_j^i are computed, such as by using a distributed Bellman-Ford algorithm (DBF) to generate a routing graph SG_j. The nodes exchange messages containing distance and status information to maintain a routing table at each node. If the distance increases for a link, or the status changes, then a diffusing computation is executed which prevents counting-to-infinity problems. Shortest path routes are selected according to loop-free invariant (LFI) conditions. The present invention solves a number of shortcomings found within current distance-vector algorithms.

An object of the invention is to provide a routing protocol for creating minimum length multipath routes within a network.

Another object of the invention is to provide a routing protocol for establishing multipath routes based on distance vectors.

Another object of the invention is to provide a method of selecting multipath routing which is not subject to loops.

Another object of the invention is to provide a method of selecting multipath routing which is not subject to counting-to-infinity problems.

Another object of the invention is to provide a routing protocol wherein the routing selections are distributed across the nodes in the given network.

Another object of the invention is to provide a multipath routing algorithm which utilizes diffusing computations to enhance performance.

Further objects and advantages of the invention will be brought out in the following portions of the specification, wherein the detailed description is for the purpose of fully disclosing preferred embodiments of the invention without placing limitations thereon.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be more fully understood by reference to the following drawings which are for illustrative purposes only:

FIG. 1 is a flowchart of the routing method according to an aspect of the present invention.

FIG. 2 is pseudocode for computing distance-vectors according to an aspect of the present invention, shown for processing both passive and active node states.

FIG. 3 is a topology diagram of the CAIRN network topology as utilized in simulations of the present invention.

FIG. 4 is a topology diagram of the MCI network topology as utilized in simulations of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

For illustrative purposes the present invention will be described with reference to FIG. 1 through FIG. 4. It will be appreciated that the apparatus may vary as to

configuration and as to details of the parts, and that the method may vary as to the specific steps and sequence, without departing from the basic concepts as disclosed herein.

The present invention provides a distance vector algorithm which is referred to herein as "Multipath Distance Vector Algorithm" (MDVA) for loop-free multipath construction.

1. Multipath Distance-Vector Algorithm (MDVA)

1.1. Solution Strategy

Given that a number of potential directed acyclic graphs (DAGs) exist for a given destination within a graph, it is problematic to determine which DAG should be utilized as a routing graph. The routing graph should be uniquely defined and it should also be easily computable by the use of a distributed algorithm. A natural choice is the use of the routing graph which is defined by the shortest paths. Accordingly, MDVA defines $S_j^i(t) = \{k \mid D_j^k(t) < D_j^i(t),\ k \in N^i\}$, where D_j^i is the cost of the shortest path from node i to node j as measured by the sum of the link-costs along the path. The routing graph SG_j implied by this set is unique and is referred to as the *shortest multipath*. In computing D_j^i, distributed routing algorithms may exchange any information, such as distance-vectors or link-states, although it must be assured that D_j^i will *converge* to the correct distances. The following formally defines what is meant as convergence.

Letting $G(t)$ denote the topology of the network as seen by an "omniscient observer" at time t, wherein $D_j^i(t)$ denotes the distance from node i to node j in $G(t)$, and

assuming that the network has a stable configuration up to a given time t. It should be noted that all quantities within G are depicted in a larger font. It can be said that the network has converged to the correct values at t if $D_j^i(t) = \hat{D}_j^i(t)$ for all i and j. If a sequence of link cost changes were to occur between time t and t_c, with none occurring subsequent to t_c, then the routing algorithm is said to converge if at some time $t_c < t_f < \infty$, $D_j^i(t_f) = \hat{D}_j^i(t_f) = D_j^i(t_c)$. In addition, during the convergence phase, the algorithm must ensure that the graph SG_j is loop-free at every instant.

According to the distributed Bellman-Ford (DBF) algorithm, each node i repeatedly executes the equation $D_j^i = min\{D_{jk}^i + l_k^i \mid k \in N^i\}$ for a given destination j and upon each D_j^i change it reports the new distance to its neighbors. A known property of DBF is the rapid rate of convergence that occurs when link costs decrease. However, convergence is not assured in the case of increasing link-costs, and when link failures result in network partitions the DBF algorithm may never converge. The lack of convergence in this instance is known in the industry as the *"counting-to-infinity problem"*. Intuitively, the counting-to-infinity problem arises as a result of "circular" logic within the distance computations, wherein a node computes its distance to a destination using a distance communicated by a neighbor, which is provided as a path-length running through the node itself. The node utilizing this distance information is unaware of the circular logic because the nodes exchange distance information and not path information.

The circular computation of distances that occur in DBF can be prevented if distance information is propagated along a DAG rooted at a destination. Given a DAG, each node computes its distance using distances reported by the "downstream" nodes and reports its distance to "upstream" nodes. This method, referred to as *diffusing computations* was first suggested by Dijkstra et. al. to ensure termination of distributed computation. It will be appreciated that a diffusion computation always terminates due to the acyclic ordering of the nodes. The base algorithm for EIGRP is DUAL which utilizes diffusing computation to solve the counting-to-infinity problem. In addition to DUAL, a number of other distance vector algorithms have been proposed which employ diffusing computations to overcome the counting-to-infinity problem of DBF. The algorithm suggested by Jaffe and Moss allows nodes to participate in multiple diffusing computations for the same destination and requires use of unbounded counters, which render the method impractical. In contrast, a node in DUAL and DASM participates in only one diffusing computation for any destination at any single time and thus requires only the use of a toggle bit. The present invention, MDVA follows the second approach.

Two issues arise regarding diffusing computation: (1) since many potential DAGs exist for a given destination, the selection of which one to use for the diffusing computation is difficult; (2) how to implement diffusing computations in a dynamic environment in which the chosen DAG changes with respect time.

The following describes resolutions for these issues. Resolving the first issue is straightforward as the shortest multipath SG_j provides a correct choice given that computing SG_j is the final objective. The resolution, however, of the second issue is

not so trivial. A routing graph SG_j utilized for carrying out a diffusing computation can be allowed to change if the following conditions are met: (1) SG_j is acyclic at *every instant*, and (2) at any given instant, if a node reports a distance through a neighbor k in S_j^i it must ensure that k remains in S_j^i until the end of the diffusing computation. The prevention of a circular computation of distances can be inferred from the following argument. Assume first that a circular computation occurs at time t involving nodes i_0, i_1, i_2, ... i_m. Let a node i_p, wherein $1 \leq p \leq m$, compute its distance at $t_p \leq t$ using distance reported by i_{p-1}, and i_0 computes its distance using the distance reported by i_m at t_0. Because i_{p-1} is held in the successor set of i_p for $1 \leq p \leq m$ and i_0 holds i_m until the diffusing computation ends, therefore it follows that:

$$i_0 \in S_j^{i_1}(t_1) \Rightarrow i_0 \in S_j^{i_1}(t)$$

$$i_1 \in S_j^{i_2}(t_2) \Rightarrow i_1 \in S_j^{i_2}(t)$$

$$\vdots$$

$$i_{m-1} \in S_j^{m}(t_m) \Rightarrow i_{m-1} \in S_j^{m}(t)$$

$$i_m \in S_j^{0}(t_0) \Rightarrow i_m \in S_j^{0}(t)$$

Because $SG_j(t)$, as implied by $S_j^i(t)$, is acyclic at every instant t, the above relations would indicate a contradiction. Thus, the circular computation is impossible when observing the above mentioned conditions. It should be noted that the distances are to be propagated along the shortest-multipath SG_j which is computed using the distances itself. This "bootstrap" approach is the core of the MDVA algorithm, which

involves computing D_j^i using diffusing computations along SG_j while simultaneously constructing and maintaining routing graph SG_j.

In order to ensure that SG_j is always loop-free a new variable *feasible distance* FD_j^i is introduced. The feasible distance FD_j^i is an "estimate" of the distance D_j^i in the sense that FD_j^i is equal to D_j^i when the network is in stable state. However, in order to prevent loops during periods of network transitions, the value of FD_j^i is allowed to differ temporarily from D_j^i. Let D_{jk}^i be the distance of k to j as notified to i by k. To ensure loop-freedom at every instant FD_j^i, D_{jk}^i, and S_j^i must satisfy the "Loop-Free Invariant" (LFI) conditions which were first introduced in regard to approximating minimum delay routing. The LFI conditions capture all previous loop-free conditions in a unified form that simplifies protocol design and correctness proofs, comprising:

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad k \in N^i \tag{1}$$

$$S_j^i(t) = \left\{k \mid D_{jk}^i(t) < FD_j^i(t)\right\} \tag{2}$$

The invariant conditions (1) and (2) state that, for each destination j, a node i can choose a successor whose distance to j, as known to i, is less than the distance of node i to j that is known to its neighbors.

Theorem 1: If the LFI conditions are satisfied at any time t, the $SG_j(t)$ implied by the successor sets $S_j^i(t)$ are loop free.

Proof: Let $k \in S_j^i(t)$ then from (2):

$$D^i_{jk}(t) \quad < \quad FD^i_j(t) \tag{3}$$

At node k, in view of node i being a neighbor and from (1) we arrive at $FD^k_j(t) \leq D^i_{jk}(t)$, which when combined with Eq. 3 yields:

$$FD^k_j(t) \quad < \quad FD^i_j(t) \tag{4}$$

It will be appreciated that Eq. 4 states that if k is a successor of node i in a path to destination j, then the feasible distance to j which is known to k is strictly less than the feasible distance of node i to j. Now, if the successor sets define a loop at time t with respect to j, then for some node p on the loop, we arrive at the absurd relation $FD^p_j(t) < FD^p_j(t)$. Therefore, the LFI conditions have been shown to be sufficient to assure loop-freedom.

The above theorem suggests that any distributed routing protocol, such as link-state or distance-vector, which attempts to determine loop-free shortest multipaths is required to compute D^i_j, FD^i_j, and S^i_j such that the LFI conditions are satisfied, and such that at convergence $D^i_j = FD^i_j$ = *minimum distance from* i *to* j.

1.2. Algorithm Description

FIG. 1 depicts the general flow for the method of the present invention. Link distances D^i_j are computed at block 10 to generate a routing graph SG_j. The nodes in the network exchange distance and status information as per block 12. If a distance increase is detected at block 14 then a diffusing computation is performed as shown in block 16. The distance and status information is used to maintain routing tables within

each node as per block 18 so that the proper selection of a loop-free route is determined according to loop-free invariant conditions as shown in block 20.

The MDVA algorithm utilizes DBF to compute distance D_j^i, and thus routing graph SG_j while always propagating distances along the routing graph SG_j to prevent counting-to-infinity problems and to otherwise ensure termination. Each node maintains a *main table* containing D_j^i as the distance of node i to destination j. The table also stores for each destination j, the successor set S_j^i, the feasible distance FD_j^i, the reported distance RD_j^i, and the shortest distance possible through the successor set S_j^i as best distance SD_j^i. In addition, the table stores $QS_j^i \subseteq S_j^i$, as the set of neighbors involved in a diffusing computation. Each node maintains a *neighbor table* for each neighbor k which contains D_{jk}^i as the distance of neighboring node k to node j as communicated by node k. A *link table* stores the link-cost l_k^i of adjacent links to each neighbor k. If a link is down its link-cost is considered to increase to infinity and the distance to unreachable nodes is also considered to be infinity.

Nodes executing the MDVA algorithm exchange information using messages containing at least one entry of the form *[type, j, d]*, where d is the distance of the node sending the message to destination j. The *type* field comprises messages such as QUERY, UPDATE, REPLY, or equivalents. It is assumed that messages transmitted over an operational link are received without errors and in the proper sequence, and that the messages are processed in the order received.

Nodes invoke the procedure *ProcessDistVect* as shown in FIG. 2 to process a

distances vector when an event occurs. An event may be considered as the arrival of a message, a change in the cost of an adjacent link, or a change in status (up/down) of an adjacent link. When an adjacent link is brought up, the node sends an update message *[*UPDATE, j, RD_j^i*]* for each destination j over the link. When an adjacent link *(*i, m*)* fails, the neighbor table associated with neighbor m is cleared and the cost of the link is set to infinity. Then for each destination, the procedure *ProcessDistVect(*UPDATE, m, ∞, j*)* is invoked. Similarly, when an adjacent link cost to m changes, the cost l_m^i, is set to the new cost and *ProcessDistVect(*UPDATE, m, D_{jm}^i, j*)* is invoked for each destination j. When a message is received, *ProcessDistVect()* is invoked for each entry of the message.

A node initializes the distance values in its tables to infinity and its sets to null at the startup time. In view of the fact that the distances can be computed independently to each destination, the remainder of the description describes the operation of the algorithm with respect to a particular destination j. A node can be in ACTIVE or PASSIVE state with respect to a destination j represented by a variable state. A node is considered active when it is engaged in a diffusing computation. Assume first that all nodes are PASSIVE. While link costs decrease, MDVA essentially operates like DBF, because the condition on line 9 always fails wherein lines 17-24 are always executed. *ProcessDistVect()* operates in such a way that when the node is in a PASSIVE state, the condition $D_j^i = FD_j^i = RD_j^i = min\left\{D_{jk}^i + l_k^i \mid k \in N^i\right\}$ always holds as can be seen from lines 8 and 23. However, if the distance to a destination increases either because the

cost of an adjacent link changes or a message is received from a neighbor, the condition on line 9 succeeds and the node engages in a diffusing computation. This is accomplished by sending query messages to all the neighbors with the best distance through the subset of neighbors S_j^i, such as SD_j^i, and waiting for the neighbors to reply (lines 14-15). The node is said to be in an ACTIVE state when it is waiting for the replies. If the increase in distance is due to a query from a successor, the neighbor is added to QS_j^i so that a reply can be given when the node transits to a PASSIVE state. When all replies are received, the node can be sure that the neighbors have the distances that the node reported and are ready to transition to the PASSIVE state. At this point, FD_j^i can be increased and new neighbors can be added to S_j^i without violating the LFI conditions.

If a query message is received from a neighbor which is not in the successor set for a node in an ACTIVE state, then a reply is given immediately. However, if the query is from a neighbor m in S_j^i, a test is performed to verify if SD_j^i increased beyond the previously reported distance, (line 28). If it did not increase beyond the limit then a reply is sent immediately. However, if SD_j^i increased, the query is blocked by adding m to QS_j^i and no reply is given. The replies to neighbors in QS_j^i are deferred until that time when the node is ready to transition to the PASSIVE state. After receiving all replies the ACTIVE phase can either end or continue. If the distance D_j^i is increased again after receipt of all replies, the ACTIVE phase will be extended by sending a new set of queries, otherwise the ACTIVE phase will terminate. For the case of ACTIVE phase

continuation, no replies are issued to the pending queries in QS_j^i. Otherwise, all replies are given and the node transits to PASSIVE state satisfying the PASSIVE state invariant $D_j^i = FD_j^i = RD_j^i = min\{D_{jk}^i + l_k^i \mid k \in N^i\}$.

2. Verifying Correctness of MDVA

The correctness of MDVA is proven for two scenarios: (1) subject to link cost decreases only, and (2) subject to some link cost increases as a result of increasing link distances. MDVA operates in a similar manner to DBF when link costs are only subject to decreases and the same proofs utilized for DBF apply. To state this formally, assume that the network is stable preceding a time t, wherein all nodes have obtained correct distances, and then at time t, the costs of a portion of the links decrease. Since the distances in the tables are such that $D_j^i(t) \geq D_j^i(t)$, within some finite time t', $t \leq t' < \infty$, and $D_j^i(t') = D_j^i(t)$. The distinction between D_j^i and D_j^i should be noted, as D_j^i is the correct distance while D_j^i is just a local variable i and is an estimate of D_j^i. It will be appreciated that by using the present routing protocol that D_j^i must eventually equal D_j^i, barring continuous changes to D_j^i.

Subject to some link cost increases, wherein distances between a portion of the source-destination pairs increase, MDVA and DBF behave differently. In this case, $D_j^i(t) < D_j^i(t)$ for some i and j. Both DBF and MDVA first increase D_j^i to a value greater than $D_j^i(t)$, after which the distances monotonically decrease until they converge to the correct distances. MDVA and DBF, however, differ on *how* they

increase the distances. DBF executes the increase step-by-step in small bounded increments until $D_j^i(t) \geq D_j^i(t)$. Unfortunately, when $D_j^i(t) = \infty$ counting-to-infinity is encountered. In contrast, MDVA executes diffusing computations to quickly raise D_j^i so that $D_j^i \geq D_j^i(t)$, after which the functioning is similar to scenario described above, and the distances converge to the correct values as before.

In summary, to show that MDVA terminates correctly, it can be shown that (1) the routing graph SG_j is loop-free at every instant; (2) every diffusing computation using routing graph SG_j completes in finite time; and (3) a finite number of diffusing computations are executed. After performing all diffusing computations the MDVA algorithm becomes similar to conventional DBF.

Theorem 2: For a given destination j, the routing graph SG_j constructed by MDVA is loop free at every instant.

Proof: The proof proceeds by illustrating that the LFI conditions are satisfied during every ACTIVE and PASSIVE phase. Let t_n be the time when the n^{th} transition to ACTIVE state starts at node i for j. The proof is by induction on t_n. At node initialization time *0*, all distance variables are initialized to infinity and hence $FD_j^i(0) \leq D_{jk}^i(0)$, and $k \in N^i$. The following is valid assuming that LFI conditions hold true up to time t_n.

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [0, t_n] \tag{5}$$

At any time t, from lines 6, 8, 14 and 23 in the pseudocode in FIG. 2, and as a result of

$SD_j^i(t) \geq D_j^i(t)$, it follows that:

$$FD_j^i(t) \leq RD_j^i(t) \quad (6)$$

and therefore, for t_{n-1} and t_n, we arrive at:

$$FD_j^i(t_{n-1}) \leq RD_j^i(t_{n-1}) \quad (7)$$

$$FD_j^i(t_n) \leq RD_j^i(t_n) \quad (8)$$

Let queries be sent at t_n, the start time of the n^{th} ACTIVE phase, to be received at a particular neighbor k at $t' > t_n$. From Eq. 6 and from the fact that if any update messages have been sent between t_{n-1} and t_0, they are non-increasing, whereby it follows that:

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t_n, t'] \quad (9)$$

The variable t'' is used to represent the time when all replies are received and the ACTIVE phase ends. During the ACTIVE phase the value of FD_j^i remains unchanged and no new RD_j^i is reported during this period (line 27-31), while during the PASSIVE phase only decreasing values of RD_j^i are reported. The following may then be derived from Eq. 8:

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t', t''] \quad (10)$$

Irrespective of whether the node transitions to the PASSIVE state or continues in the ACTIVE phase, at time t'' the following is known from Eq. 6:

$$FD_j^i(t'') \leq RD_j^i(t'') \quad (11)$$

In the case that the ACTIVE phase finally terminates, we arrive at $FD_j^i(t) \leq D_{jk}^i(t)$ for $t \in [t_n, t'']$. In the PASSIVE state, RD_j^i is can only decrease until the next ACTIVE phase at t_{n+1}. Therefore, the LFI conditions are satisfied in the interval $[t_n, t_{n+1}]$. Alternatively, if the ACTIVE state continues then new queries are sent at t''. Assuming that all replies for these queries are received at t''', and from a similar argument as above, it follows that $FD_j^i(t) \leq D_{jk}^i(t)$ for $t \in [t_n, t''']$. It will be appreciated, therefore, that irrespective of the duration of the ACTIVE phase the invariant holds between the times $[t_n, t_{n+1}]$. As a consequence of which, by induction the LFI conditions hold at all times. It follows from Theorem 1 that routing graph SG_j is loop-free at all times.

Lemma 1: Every ACTIVE phase is subject to a finite duration.

Proof: An ACTIVE phase may never end due to either "deadlock" or "livelock". It will be recognized that a node transitioning to the ACTIVE state, with respect to a given destination, will transmit queries. If the transition occurs as a result of a query from a successor, the node defers the reply to this query until it receives the replies to its own queries. An issue of "circular" waits arises as a consequence of nodes awaiting replies to their own queries before replying to a query from a neighbor. It should be recognized that "circular" waits can lead to deadlock conditions. However, in the present invention "circular" waits are prevented for the following reasons. Firstly, a node in the passive state immediately replies to a query from a predecessor (lines 19). If the query is from a successor that potentially increases SD_j^i, and the node is ACTIVE, the query is held until the ACTIVE phase ends (line 29). As a result of the routing graph

SG_j being loop-free at every instant, as illustrated by the proof to Theorem 2, a deadlock condition cannot occur. Thus a node issuing queries to its neighbors will eventually receive all the replies and transition to the PASSIVE state.

A livelock is a situation in which a node endlessly has continuous back-to-back ACTIVE phases without ever being able to reply to the pending queries from its successors. It will be appreciated that a livelock also is not possible within the present system for the following reasons. An ACTIVE phase transition occurs either because of a query from a successor or a link-cost increase of an adjacent link. A query from a successor is blocked if it increases best distance SD_j^i. Since links can change only a finite number of times and a finite number of neighbors exist for each node from which the node can receive queries, the node can only enter a finite number of back-to-back active phases. A node eventually sends all pending replies and enters the PASSIVE state, wherein livelock is not possible.

Lemma 2: A node can have only a finite number of ACTIVE phases.

Proof: It is assumed for the sake of contradiction that a node *does* exist which proceeds through an infinite number of PASSIVE to ACTIVE transitions. An active phase transition occurs either because of a query from a successor or a link-cost increase of an adjacent link. The infinite PASSIVE-ACTIVE phase transitions must be triggered by an infinite number of queries from a neighbor, because link costs can change only a finite number times. Let that neighbor be represented by node k. Now, by the same argument, node k is sending infinite queries because it is receiving infinite queries. However, this argument cannot be continued indefinitely because there are

only finite number of nodes in the network. Since the reply to the neighbor in the successor set causing the phase transition is blocked, and the routing graphs are loop-free at every instant (Theorem 2), there must exist a node that transitions to the ACTIVE state only because of adjacent link cost changes. This implies a link changes cost an infinite number of times which is a contradiction of the assumption, which proves that a node cannot have infinite ACTIVE phases.

Theorem 3: After a finite sequence of link-cost changes in the network, the distances D_j^i converge to the final correct values D_j^i.

Proof: Assume at time *0* that every node has correct values for all link distances. In other words, $D_j^i(0) = D_j^i(0)$. Assume a finite number of link cost changes, link failures and link recoveries occurring in the network between time *0* and time t_c, and after time t_c that no additional changes occur. It must be shown that at some time t_f, such that $t_c \leq t_f \leq \infty$, wherein all nodes converge to the correct distances given by $D_j^i(t_f) = D_j^i(t_c) = D_j^i(t_f)$.

From Lemma 1 and 2, it follows that all nodes, within a finite time after the last link change will transition to the PASSIVE state and remain in PASSIVE state thereafter. Therefore, let t' be the time when the last ACTIVE phase ends in the network, wherein the following are to be proven.

1. $D_j^i(t') \geq D_j^i(t_c)$ for every i and j.

2. In the time period between time t' and time t_f, every distance D_j^i

monotonically decreases and eventually converges at time t_f to the correct distances $D_j^i(t_c)$. Wherein $D_j^i(t_f) = D_j^i(t_c)$.

Proof, Part 1: Assume towards a contradiction that $D_j^i(t') < D_j^i(t_c)$. Let $D_j^i(t') = (l_k^i(t') + D_{jk}^i(t'))$ for some $k \in K \subseteq N^i$. Assume $D_j^k(t') \leq D_j^k(t_c)$, and that K has only one element. Because $D_j^i(t_c) = l_k^i(t_c) + D_j^k(t_c)$ we have $l_k^i(t') + D_{jk}^i(t') \leq l_k^i(t_c) + D_j^k(t')$ from which we can infer that either $l_k^i(t') < l_k^i(t_c)$ or $D_{jk}^i(t') < D_j^k(t')$ or both. If $l_k^i(t') < l_k^i(t_c)$, it implies that the link cost of (i, k) is not yet increased to $l_k^i(t_c)$ via a link-cost change event. When it does, the condition on line 9 becomes true and an ACTIVE state transition is triggered, and all ACTIVE phases have not terminated. Similarly, if $D_{jk}^i(t') < D_j^k(t')$, then messages are in-transit that when processed by node i would trigger a PASSIVE-to-ACTIVE transition. Thus, the ACTIVE phases have not ended, which contradicts the original erroneous assumption. Therefore, when ACTIVE phases end $D_j^i(t') \geq D_j^i(t_c)$. When K has more than one element, each element will be sequentially removed from the successor set without triggering the ACTIVE transition until the last element, at which time the ACTIVE state transition finally occurs.

Proof Part 2: After every node becomes PASSIVE at time t', all the messages in-transit can only decrease the distances; otherwise, that would result in a transition to an ACTIVE state. At this stage MDVA works essentially like DBF and the same proof of

DBF applies here. Each time a distance is decreased, the new distance is reported. The distances will eventually converge, because distances cannot decrease forever and are bounded on the lower end by $D_j^i(t_c)$.

3. Evaluating the Performance of MDVA

The *storage complexity* is determined by the amount of table space needed by any given node. Each one of the N^i neighbor tables and the main distance table has size of the order $O(|N^i||N|)$. The storage complexity is, therefore, of the order $O(|N|)$. The *computation complexity* is the time taken to process a distance vector and it is easy to see that *processDistVector()* requires execution time given by $O(|N^i|)$. The *time complexity* is the time it takes for the network to converge after a set of link-cost changes occur within the network. The *communication complexity* is the amount of message overhead required for propagating a set of link-cost changes. In a dynamic environment, the timing and range of link-cost changes occur in complex patterns and is often determined by the nature of the traffic on the network. Thus, obtaining expressions for time complexity and communication complexity in closed form is not possible, and only approximations are provided for the case in which communication is synchronous throughout the network.

Accordingly, simulations are utilized to compare the worst case performance, in terms of control overhead and convergence times, of MDVA with those of DBF and MPATH. The purpose of these simulations is to yield qualitative explanations for the behavior and performance of MDVA. The reason for choosing DBF as a benchmark is

that it does not use diffusing computations and yet is based on vectors of distances. The reason for choosing MPATH is that it has been shown to be very efficient, in terms of communication overhead and convergence times, compared against prior algorithms based on link-state information and distance information, such topology broadcast, DASM, LVA, ALP. Thus DBF and MPATH represent two ends of the performance spectrum.

MDVA achieves loop-freedom through diffusing computations that, in some cases, may span the whole network. In contrast, MPATH uses only neighbor-to-neighbor synchronization. It is interesting to see how convergence times are effected by the synchronization mechanisms. Also, it is not obvious how the control message overheads of MDVA and MPATH compare.

The performance metrics used for comparison are the control message overhead and the convergence times. It is assumed that the computation times are negligible in relation to the communication times. The simulator utilized was an event-driven real-time simulator called CPT. Simulations are performed on the CAIRN and MCI topology shown in FIG. 3 and FIG. 4 respectively. The bandwidth and propagation delays of each link are given in parenthesis next to the topology. In backbone networks the links and nodes are highly reliable and change status much less frequently than link costs which are a function of the traffic on the link. This is particularly true when near-optimal delay routing is utilized, in which the link costs are periodically measured and reported. For these reasons, the algorithms are compared when multiple link-cost changes occur. Link costs are chosen randomly within a range and link-cost change events are

triggered, at which time the algorithms are allowed to converge. The worst case message overhead and convergence times are shown in Table 2 and Table 3 respectively. MDVA provides a performance increase over DBF by virtue of the utilization of diffusing computations for increasing distances. MPATH was found to achieve higher performance than MDVA in the majority of instances, although, at times MDVA outperformed MPATH as can be seen for MCI*(0.1mS, 10Mb)*, which generally occurs when link-cost changes are largely link decreases as distance-vector algorithms are known to converge rapidly when link-costs decrease.

Accordingly, it will be seen that this invention presents a new distributed distance-vector routing algorithm which provides multiple next-hop choices for each destination wherein the routing graphs implied by the multiple next-hop choices are always loop-free. The present invention utilizes a set of loop-free invariant conditions that ensure correct termination of the algorithm and eliminate counting-to-infinity problems. The multiple successors that MDVA makes available at each node can be used for traffic load-balancing. It has been shown utilizing other known algorithms, such as MPDA, that loop-free multiple paths are necessary in order to minimize the delays encountered within the network. It will be appreciated, therefore, that MDVA can be utilized as an alternative to MPDA to approximate minimum-delay routing in networks.

Although the description above contains many specificities, these should not be construed as limiting the scope of the invention but as merely providing illustrations of some of the presently preferred embodiments of this invention. Therefore, it will be appreciated that the scope of the present invention fully encompasses other

embodiments which may become obvious to those skilled in the art, and that the scope of the present invention is accordingly to be limited by nothing other than the appended claims, in which reference to an element in the singular is not intended to mean "one and only one" unless explicitly so stated, but rather "one or more." All structural, chemical, and functional equivalents to the elements of the above-described preferred embodiment that are known to those of ordinary skill in the art are expressly incorporated herein by reference and are intended to be encompassed by the present claims. Moreover, it is not necessary for a device or method to address each and every problem sought to be solved by the present invention, for it to be encompassed by the present claims. Furthermore, no element, component, or method step in the present disclosure is intended to be dedicated to the public regardless of whether the element, component, or method step is explicitly recited in the claims. No claim element herein is to be construed under the provisions of 35 U.S.C. 112, sixth paragraph, unless the element is expressly recited using the phrase "means for."

Table 1

Reference for Notations

N	Set of nodes in the network
N^i	Set of neighbors for node i
S_j^i	Subset of N^i that node i forwards packets of destination j
SG_j	Routing graph implied by the successor sets of destination j
D_j^i	Distance of node i to node j as known to node i
l_k^i	Cost of link (i, k)
D_{jk}^i	Distance of node k to j as reported to node i by node k
FD_j^i	Feasible distance is an estimate of D_j^i
RD_j^i	Distance to j as reported by node i to its neighbors
SD_j^i	Best distance to j through S_j^i
QS_j^i	Set of neighbors that are awaiting replies
$G(t)$	An overview of the network at time t
$D_j^i(t)$	Distance of node i to node j in $G(t)$
$l_k^i(t)$	Cost of link (i, k) in $G(t)$

Table 2

Overhead Loading

	DBF	MDVA	MPATH
Topology and conditions	Message Load (bits)		
MCI (10mS, 10Mb)	62568	52352	32408
MCI (0.1mS, 10Mb)	78624	52840	32408
CAIRN (10mS, 10Mb)	39648	14056	6176
CAIRN (0.1mS, 10Mb)	37208	12992	5640

Table 3

Convergence Times

	DBF	MDVA	MPATH
Topology and conditions	Conversion Time in milliseconds (mS)		
MCI (10mS, 10Mb)	330.51	250.46	190.72
MCI (0.1mS, 10Mb)	4.36	2.51	2.62
CAIRN (10mS, 10Mb)	470.61	170.31	150.32
CAIRN (0.1mS, 10Mb)	4.07	2.14	1.82